Course: Programing concepts in scientific programming

Final project

Project 1: Linear systems

Introduction:

The aim of the project is to implement direct and iterative methods to solve linear systems. To do so we will implement a C++ program. In this report we will present the methods, we will show you how to compile the program, its typical execution, some of its features, some validation tests and presents issues and perspectives.

Methods:

A linear system is a system of n linear equation that can be written in the following form:

$$Ax = b$$

With:

- A: a matrix nXn of coefficient multiplying the variables of the problem
- x: a vector nx1 of the variables of the problem
- b: a vector nx1 containing the right-hand side of the equations

We present different methods that we will implement in the program to solve linear system. Two are directs methods, LU and Cholesky, meaning that they will lead to exact solution for defined systems (neglecting numerical rounding). Four other methods are iterative: Conjugate gradient, Jacobi, Gauss-Seidel and Richardson. This means that the solution is reached only if the algorithm converges.

For direct methods:

LU “lower upper” decomposition: the idea is to decompose the A matrix into the product of a upper triangular and a lower triangular matrix:

$$PA = LU$$

Once the decomposition is known the problem is easily solved as two sub problem.

Cholesky decomposition: the algorithm is applicable to positive definite matrix. The decomposition of A is of the form:

$$PA = L^T L$$

The decomposition is faster than the LU. The rest of the process is the same as the other methods.

For iterative methods:

For an iterative method, the idea is to guess an initial solution and evaluate it. Then we iterate until we find a solution that satisfies the initial problem. The algorithm either converges to the solution or diverge from it. It is needed to use a convergence criteria and a stopping criteria. The different methods vary in the way that they estimate the next iteration step. The algorithm then has different convergence criteria.

Algorithm structure:

The program structure consists of a main function where the different solving algorithm are called.

The general form of the problem is defined in the abstract class linSys. In this class, the Matrix A and Vector b are defined. The linear system has then two subclasses the Exact class or Approx class. The Exact and Appox class don't have the same parameter definition. For the Approx class, we define tolerance, initial guess and max number of iteration. LU and Cholesky are subclasses of the Exact class and CG, GS, Jacobi and Richardson are subclasses of Approx. In each of these subclasses, the different methods to solve the linear system are implemented.

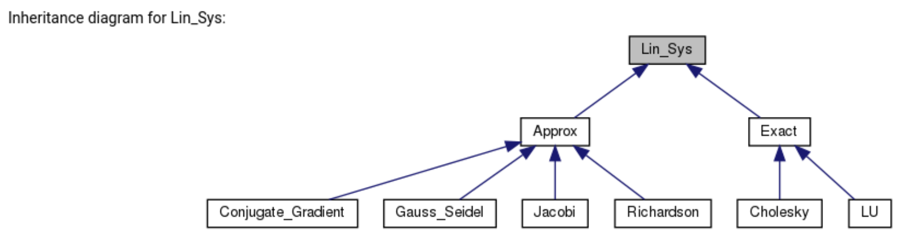


Figure 1: Structure of the linear system abstract class

The program needs to do an important number of operations with matrix and vector. Therefore, we implemented one Vector class and one Matrix class in order to simplify the implementation.

Run the program:

The user can run the program without input files. He can then run from the console using the Makefile or from an IDE. To give input data as data files the program should be run from the console.

Running the code from IDE (clion):

When running the code, it is needed to give first the size of the problem as an integer, and then the value of matrix A and vector b.

After giving the initial values in the command, the following command is shown:

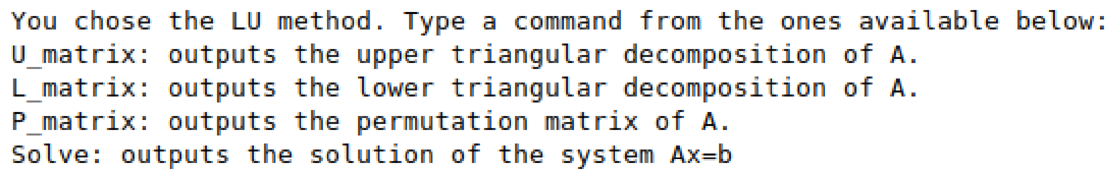
```
Description of available methods:
----------------------
------------------------
Exact : (suitable for relatively small systems)
---------------------------------------------
Cholesky : Fast but for semipositive definite systems only.
LU : Works for all definite systems.

Approximative : (suitable for large systems)
----------------------------------------
Conjugate_gradient : very fast
Gauss_Seidel : Works best for diagonally dominant systems.
Jacobi : Fast but for semipositive definite systems and diagonally dominant systems.
Richardson : Works for semipositive definite systems.


Choose a method by typing a string corresponding to the name of the method, or Exit to abort program
Conjugate_gradient
```

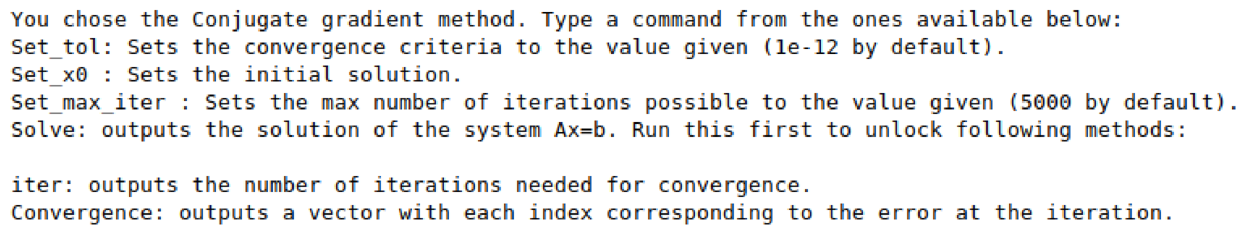
You can type the name of the method in order to execute it. The algorithm then execute the method "run_algo" with the following inputs: (A,b, *string typed in command*).

When choosing an exact method, for example LU, the following text appears:



```
You chose the LU method. Type a command from the ones available below:
U_matrix: outputs the upper triangular decomposition of A.
L_matrix: outputs the lower triangular decomposition of A.
P_matrix: outputs the permutation matrix of A.
Solve: outputs the solution of the system Ax=b
```

The text is different if the method is approximate:



```
You chose the Conjugate gradient method. Type a command from the ones available below:
Set_tol: Sets the convergence criteria to the value given (1e-12 by default).
Set_x0 : Sets the initial solution.
Set_max_iter : Sets the max number of iterations possible to the value given (5000 by default).
Solve: outputs the solution of the system Ax=b. Run this first to unlock following methods:

iter: outputs the number of iterations needed for convergence.
Convergence: outputs a vector with each index corresponding to the error at the iteration.
```

It is possible for the user to set the tolerance, the initial guess and the number of iteration.

When the user types “solve” the code outputs the solution x on the console. To get the ouput as a .dat file. It is needed to run the program in the terminal and give correct input files. It would be possible to add a feature to write a .dat file with the solution.

When typing “Exit” the user can then use another method to solve the linear system he defined.

To exit the program, write “Exit” when the algorithm offers to choose a method.

Running the code from the terminal:

The first command to type the make all command

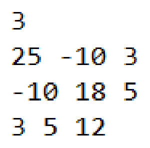
It is needed to run the program and give it the input. The command for exact methods has 4 terms:

./Linear_System A.dat b.dat LU

For the approximate methods, the number of argument can be up to 7:

./Linear_System A.dat b.dat Gauss_Seidel x_0.dat tol.dat max_iter

The input needs to be formatted as follow:



```
3
25 -10 3
-10 18 5
3 5 12
```

Figure 2 : Exemple for a 3x3 matrix

The first term specifies the size of the system and on the next lines, the matrix or vector are written with a space between each terms.

Then the program works the same as in the IDE.

When the solution is found, the command make clean is used to delete both the .o files the executables and the data files.

List of features of the program:

The list of features is presented in the screenshots for the direct and approximate methods.

- For each direct method: L_Matrix, U_Matrix, P_matrix, Solve.
- For each approximate method: Set_tol, Set_x0, Set_max_iter, Solve, converge, iter.

In total it represents 32 features.

Google tests:

In total 14 tests are executed. The solution for two linear systems, are evaluated for each method. In each case we compare the expected solution with solution calculated in Matlab. The all tests succeeded as all algorithm converges to the solution. We use the command ASSERT_NEAR since the results are not strictly identical. We also ran tests in order to test that the class Matrix and Vector works properly by doing vector and matrix operations.

Limitations and problems:

Inputs: We implemented two ways to give the input of the problem, one using files and the second using the command interface. To work in our program, the files need to be precisely formatted. It needs to be .dat files and the size of the problem needs to be stated. This is not the best and one way to go around it would be for the program to evaluate the size of the input. We could also implement method to read for different types of files.

The program doesn't handle all input errors: give a double for the size of the problem, or put letters instead of number in the matrix or vectors.

Outputs: we could add a feature to write in a .dat file the solution of the problem when solving using the console interface. Again we could propose to have the solution in different format. Also, the user doesn't have an influence on the precision of the outputs.

The program: all the resolution methods were implemented. We could add some features like preconditioners for the conjugate gradient. For the Jacobi and Gauss Seidel over relaxation method could help those algorithms to converge faster to the solution.

We implemented convergence checks to help the user to choose the best method. This checks could be improved and be more general.